```
pip install  dgl -f https://data.dgl.ai/wheels/torch-2.3/cu121/repo.html
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import dgl
import os
from dgl.nn import GraphConv, HeteroGraphConv
import numpy as np
import torch.optim as optim
from torch.optim import Adam
from sklearn.model_selection import train_test_split
from torch.utils.data import Dataset
from dgl.dataloading import GraphDataLoader
```

```
DGL backend not selected or invalid.  Assuming PyTorch for now.
Setting the default backend to "pytorch". You can change it in the ~/.dgl/config.json file or export the DGLBACKEND environment variable.  Valid options are: pytorch, mxnet, tensorflow (all lowerc
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print('Using device:', device)
```

```
Using device: cuda
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
dirve_path = '/content/drive/MyDrive/Project/'
train = pd.read_csv(dirve_path + 'security_train.csv')
```

```
api_to_idx = {api: idx for idx, api in enumerate(train['api'].unique())}
train['api_idx'] = train['api'].map(api_to_idx)
train.sort_values(by=['file_id', 'index'], inplace=True)
```

```
train
```

	file_id	label	api	tid	index	api_idx
0	1	5	LdrLoadDll	2488	0	0
4744	1	5	LdrLoadDll	2572	0	0
5108	1	5	LdrLoadDll	2648	0	0
1	1	5	LdrGetProcedureAddress	2488	1	1
4745	1	5	LdrGetProcedureAddress	2572	1	1
...	...	...	...	...	...	...
89806688	13887	2	NtClose	2336	618	3
89806689	13887	2	NtClose	2336	619	3
89806690	13887	2	NtClose	2336	620	3
89806691	13887	2	NtClose	2336	621	3
89806692	13887	2	NtTerminateProcess	2336	622	57

89806693 rows × 6 columns

```
# Create mapping of file IDs and thread IDs to indexes
file_ids = pd.unique(train['file_id'])
thread_ids = pd.unique(train['tid'])
file_mapping = {fid: i for i, fid in enumerate(file_ids)}
thread_mapping = {tid: i for i, tid in enumerate(thread_ids)}
```

```
file_ids = train['file_id'].unique()
train_file_ids, val_file_ids = train_test_split(file_ids, test_size=0.2, random_state=42)
train_data = train[train['file_id'].isin(train_file_ids)]
val_data = train[train['file_id'].isin(val_file_ids)]
```

```
def create_hetero_graphs(df, api_to_idx):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    file_ids = df['file_id'].unique()
    batched_graphs = []

    # Pre-calculated API and index mapping of threads
    apis_indices =  api_to_idx
    tid_indices = thread_mapping
    for file_id in file_ids:
        sub_df = df[df['file_id'] == file_id].reset_index(drop=True)

        # Graph initialisation
        num_apis = len(apis_indices)
        num_threads = len(tid_indices)
        g = dgl.heterograph({
            ('file', 'calls', 'api'): ([], []),
            ('api', 'executed_in', 'thread'): ([], [])
        }, num_nodes_dict={
            'file': 1, 'api': num_apis, 'thread': num_threads
        }, device=device)

        # Add node features
        g.nodes['file'].data['field_id'] = torch.tensor([[file_id]], dtype=torch.float, device=device)
        g.nodes['api'].data['api_idx'] = torch.tensor(list(apis_indices.values()), dtype=torch.long, device=device)
        g.nodes['thread'].data['tid'] = torch.tensor(list(tid_indices.values()), dtype=torch.long, device=device)
        label = sub_df['label'].iloc[0]  # Get tags associated with the entire file
        # File nodes added to the graph
        g.nodes['file'].data['label'] = torch.tensor([[label]], dtype=torch.long, device=device)

        # Optimisation: using NumPy to compute edges directly
        calls_src = np.zeros(len(sub_df), dtype=int)  # The index of the file node is always 0
        calls_dst = np.array([apis_indices[api] for api in sub_df['api']])
        exec_src = calls_dst  # API to threads with same source node and calling relationship
        exec_dst = np.array([tid_indices[tid] for tid in sub_df['tid']])

        # Add edges
        g.add_edges(torch.tensor(calls_src, device=device), torch.tensor(calls_dst, device=device), etype='calls')
        g.add_edges(torch.tensor(exec_src, device=device), torch.tensor(exec_dst, device=device), etype='executed_in')

        batched_graphs.append(g)

    batched_graph = dgl.batch(batched_graphs)
    return batched_graph
```

```
train_graph = create_hetero_graphs(train_data, api_to_idx)
```

```
val_graph = create_hetero_graphs(val_data, api_to_idx)
```

```
val_graph
```

```
Graph(num_nodes={'api': 819510, 'file': 2778, 'thread': 7728396},
      num_edges={('api', 'executed_in', 'thread'): 16957093, ('file', 'calls', 'api'): 16957093},
      metagraph=[('api', 'thread', 'executed_in'), ('file', 'api', 'calls')])
```

```
train_graph
```

```
Graph(num_nodes={'api': 3277155, 'file': 11109, 'thread': 30905238},
      num_edges={('api', 'executed_in', 'thread'): 72849600, ('file', 'calls', 'api'): 72849600},
      metagraph=[('api', 'thread', 'executed_in'), ('file', 'api', 'calls')])
```

```
train_graphs_filename = '/content/drive/My Drive/Graphs/train_hetero_graphs.bin'
val_graphs_filename = '/content/drive/My Drive/Graphs/val_hetero_graphs.bin'
```

```
def save_graphs(graph, filename):
    dgl.save_graphs(filename, [graph])
```

```
save_graphs(train_graph, train_graphs_filename)
```

```
save_graphs(val_graph, val_graphs_filename)
```

```
def load_graphs_from_file(filename):
    graphs, _ = dgl.load_graphs(filename)
    return graphs
```

```
train_graphs = load_graphs_from_file(train_graphs_filename)
val_graphs = load_graphs_from_file(val_graphs_filename)
```

```
train_graphs = train_graphs[0]
```

```
train_graphs
```

```
Graph(num_nodes={'api': 3277155, 'file': 11109, 'thread': 30905238},
      num_edges={('api', 'executed_in', 'thread'): 72849600, ('file', 'calls', 'api'): 72849600},
      metagraph=[('api', 'thread', 'executed_in'), ('file', 'api', 'calls')])
```

```
val_graphs = val_graphs[0]
val_graphs
```

```
Graph(num_nodes={'api': 819510, 'file': 2778, 'thread': 7728396},
      num_edges={('api', 'executed_in', 'thread'): 16957093, ('file', 'calls', 'api'): 16957093},
      metagraph=[('api', 'thread', 'executed_in'), ('file', 'api', 'calls')])
```

```
loaded_graphs
```

```
[Graph(num_nodes={'api': 3277155, 'file': 11109, 'thread': 30905238},
       num_edges={('api', 'executed_in', 'thread'): 72849600, ('file', 'calls', 'api'): 72849600},
       metagraph=[('api', 'thread', 'executed_in'), ('file', 'api', 'calls')])]
```

```
class GraphDataset(Dataset):
    def __init__(self, graphs, labels):
        self.graphs = graphs  # It's a list of diagrams
        self.labels = labels  # It's a list of tags that correspond to the diagram #

    def __len__(self):
        return len(self.graphs)

    def __getitem__(self, idx):
        return self.graphs[idx], self.labels[idx]
```

```
print(type(train_graphs))
```

```
<class 'dgl.heterograph.DGLGraph'>
```

```
individual_train_graphs = dgl.unbatch(train_graphs)
train_labels = [g.ndata['label']['file'][0].item() for g in individual_train_graphs]
```

```
individual_val_graphs = dgl.unbatch(val_graphs)
val_labels = [g.ndata['label']['file'][0].item() for g in individual_val_graphs]
```

```
train_dataset = GraphDataset(individual_train_graphs, train_labels)
train_loader = GraphDataLoader(train_dataset, batch_size=10, shuffle=True, drop_last=False)
```

```
val_dataset = GraphDataset(individual_val_graphs, val_labels)
val_loader = GraphDataLoader(val_dataset, batch_size=10, shuffle=True, drop_last=False)
```

```
class HeteroRGCN(nn.Module):
    def __init__(self, num_apis, num_threads, embedding_dim, hidden_dims, num_classes):
        super(HeteroRGCN, self).__init__()
        self.api_embedding = nn.Embedding(num_apis, embedding_dim)
        self.thread_embedding = nn.Embedding(num_threads, embedding_dim)

        self.conv1 = HeteroGraphConv({
            'calls': GraphConv(embedding_dim, hidden_dims),
            'executed_in': GraphConv(embedding_dim, hidden_dims)
        }, aggregate='sum')

        self.conv2 = HeteroGraphConv({
            'calls': GraphConv(hidden_dims, hidden_dims),
            'executed_in': GraphConv(hidden_dims, hidden_dims)
        }, aggregate='sum')

        self.classify_thread = nn.Linear(hidden_dims, num_classes)

    def forward(self, g, api_ids, thread_ids):
        g.nodes['api'].data['api_idx'] = self.api_embedding(api_ids)
        g.nodes['thread'].data['tid'] = self.thread_embedding(thread_ids)

        h_dict = self.conv1(g, {'api': g.nodes['api'].data['api_idx'], 'thread': g.nodes['thread'].data['tid']})
        #h_dict = self.conv1(g, h)
        h_dict = {k: F.relu(v) for k, v in h_dict.items()}

        #h_dict = self.conv2(g, h_dict)
        #h_dict = {k: F.relu(v) for k, v in h_dict.items()}

        # Ensure that each node type is correctly updated
        for ntype in h_dict:
            g.nodes[ntype].data['h'] = h_dict[ntype]
            print(f"Updated {ntype} features with shape: {h_dict[ntype]}") # Print feature shapes

        if 'thread' in h_dict:
            thread_repr = dgl.mean_nodes(g, 'h', ntype='thread')
            thread_out = self.classify_thread(thread_repr)
            return thread_out
        else:
            print("No thread type features available.")  # If there is no thread type feature, print the prompt
            return None
```

```
num_apis = len(api_to_idx)
num_files = len(file_mapping)
num_threads = len(thread_mapping)
embedding_dim = 8
hidden_dims = 16
num_classes = 8

model = HeteroRGCN(num_apis, num_threads, embedding_dim, hidden_dims, num_classes)
```

```
def train(model, train_loader, val_loader, optimizer, criterion, device, num_epochs, patience):
    model.to(device)
    best_val_loss = np.inf
    epochs_no_improve = 0  # Used to follow up to verify that losses have stopped improving

    for epoch in range(num_epochs):
        model.train()
        total_loss = 0

        for batched_graph, labels in train_loader:
            batched_graph = batched_graph.to(device)
            labels = labels.to(device)

            # Clear the gradient
            optimizer.zero_grad()

            # Because it is a heterogeneous graph, we need to pass the ID corresponding to the node type
            api_ids = batched_graph.nodes['api'].data['api_idx'].to(device).long()
            thread_ids = batched_graph.nodes['thread'].data['tid'].to(device).long()
            #file_ids = batched_graph.nodes['file'].data['field_id'].to(device).long()
            print(api_ids)
            print(thread_ids)
            # print(file_ids)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
num_epochs = 50
patience = 20
            loss.backward()

torch.cuda.empty_cache()


train(model, train_loader, val_loader, optimizer, criterion, device, num_epochs, patience)
```

```
tensor([  0,   1,   2,  ..., 292, 293, 294], device='cuda:0')
tensor([   0,    1,    2,  ..., 2779, 2780, 2781], device='cuda:0')
Updated thread features with shape: tensor([[0.0000, 0.2055, 0.0000,  ..., 1.1592, 2.8373, 0.0000],
        [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.3702, 0.2462],
        [0.0000, 0.6761, 0.0000,  ..., 0.0043, 1.0790, 0.0000],
        ...,
        [0.0000, 0.0000, 0.0000,  ..., 0.0176, 0.0175, 0.0175],
        [0.0000, 0.0000, 0.0000,  ..., 0.0176, 0.0175, 0.0175],
        [0.0000, 0.0000, 0.0000,  ..., 0.0176, 0.0175, 0.0175]],
       device='cuda:0', grad_fn=<ReluBackward0>)
Updated thread features with shape: tensor([[0.0114, 0.0238, 0.0000,  ..., 0.0000, 0.0000, 0.0482],
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186],
        [0.0114, 0.0238, 0.0000,  ..., 0.0000, 0.0000, 0.0482],
        ...,
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186],
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186],
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186]],
       device='cuda:0')
Epoch 19/50, Train Loss: 1.9996, Val Loss: 1.9546
tensor([  0,   1,   2,  ..., 292, 293, 294], device='cuda:0')
tensor([   0,    1,    2,  ..., 2779, 2780, 2781], device='cuda:0')
Updated thread features with shape: tensor([[0.0000, 0.1992, 0.0000,  ..., 1.1736, 2.8557, 0.0000],
        [0.0000, 0.0000, 0.0000,  ..., 0.0000, 0.3745, 0.2492],
        [0.0000, 0.6729, 0.0000,  ..., 0.0068, 1.0858, 0.0000],
        ...,
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186],
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186],
        [0.0000, 0.0000, 0.0000,  ..., 0.0187, 0.0185, 0.0186]],
       device='cuda:0', grad_fn=<ReluBackward0>)
Updated thread features with shape: tensor([[0.0102, 0.0226, 0.0000,  ..., 0.0000, 0.0000, 0.0496],
        [0.0000, 0.0000, 0.0000,  ..., 0.0197, 0.0195, 0.0196],
        [0.0102, 0.0226, 0.0000,  ..., 0.0000, 0.0000, 0.0496],
        ...,
        [0.0000, 0.0000, 0.0000,  ..., 0.0197, 0.0195, 0.0196],
        [0.0000, 0.0000, 0.0000,  ..., 0.0197, 0.0195, 0.0196],
        [0.0000, 0.0000, 0.0000,  ..., 0.0197, 0.0195, 0.0196]],
       device='cuda:0')
Epoch 20/50, Train Loss: 1.9977, Val Loss: 1.9559
```